HydEE : Vers un protocole de recouvrement arrière hiérarchique pour les machines exascales De l'exploitation du déterminisme des émissions dans les protocoles de recouvrement arrière

Thomas Ropars, Amina Guermouche, Franck Cappello

HydEE : Vers un protocole de recouvrement arrière hiérarchique pour les machines exascales

De l'exploitation du déterminisme des émissions dans les protocoles de recouvrement arrière

Amina Guermouche* — **Thomas Ropars**** — **Franck Cappello*****

* *Université Paris Sud, Orsay, France*
guermou@lri.fr

** *École Polytechnique Fédérale de Lausanne, Lausanne, Suisse*
thomas.ropars@epfl.ch

*** *INRIA Saclay-Île de France,*
University of Illinois at Urbana-Champaign, Urbana, IL, USA
fci@lri.fr

RÉSUMÉ. *Dans la perspective de la conception de super-calculateurs exascale, de nouvelles solutions de tolérance aux fautes doivent être trouvées. Pour les applications parallèles à échange de messages, les protocoles de recouvrement arrière existants ne sont pas adaptés. Pour pouvoir supporter des applications de très grande taille et des fréquences de défaillance élevées, un protocole doit être capable de confiner les conséquences des défaillances à un petit sous ensemble de processus tout en offrant de bonnes performances en fonctionnement normal, et en limitant les quantités de données à sauvegarder, en particulier en mémoire. Pour répondre à ces objectifs, nous proposons HydEE, un protocole de recouvrement arrière hiérarchique combinant sauvegarde de points de reprise coordonnés et enregistrement de messages. HydEE se fonde sur le déterminisme des émissions des applications parallèles du calcul scientifique pour pouvoir tolérer des fautes multiples sans recourir à un support de stockage stable. Nos évaluations montrent que pour la plupart des applications, sauvegarder le contenu de moins de 15% des messages en mémoire, permet de limiter les retours arrière après une défaillance à moins de 15% des processus.*

ABSTRACT. *The move towards exascale super-computers requires new fault tolerance solutions. Regarding parallel message passing applications, existing rollback-recovery protocols are not suited. To be able to deal with very large scale applications and high failure rate, a proto-*

col should be able to confine failures consequences to a small subset of the processes, while providing good failure free performance, and logging a limited amount of data, especially in memory. To fulfill these needs, we propose HydEE, a hierarchical rollback-recovery protocol that combines coordinated checkpointing and message logging. HydEE leverages the send-determinism of scienfitic parallel applications to tolerate multiple failures without relying on a stable storage. Our experiments show that for most applications, saving less than 15% of the messages payload in memory is enough to limit the rollbacks after a failure to less than 15% of the processes.

MOTS-CLÉS : *Calcul haute performances, applications parallèles, MPI, tolérance aux fautes, recouvrement arrière, protocoles hiérarchiques*

KEYWORDS: *High performance computing, parallel applications, MPI, fault tolerance, rollback-recovery, hierarchical protocols*

1. Introduction

Pour répondre aux besoins en capacité de calcul des applications de calcul scientifique, des super-calculateurs toujours plus puissants sont construits. Aujourd'hui, le super-calculateur le plus puissant du monde atteint 8 petaflops (nombre d'opérations à virgule flottante par seconde), et comprend plus de 500000 cœurs de calcul (www.top500.org, 2011). Les études ont déjà commencé pour concevoir la première machine qui attendra l'exaflop (10^{18} flops) à l'horizon 2020 (Kogge *et al.*, 2008). À une telle échelle, les défaillances ne peuvent plus être considérées comme rares, et des mécanismes de tolérance aux fautes deviennent indispensables pour assurer la bonne terminaison des applications. Le temps moyen entre deux défaillances attendu pour ces systèmes est de l'ordre de quelques heures (Elnozahy *et al.*, 2008).

Les applications parallèles exécutées sur ce type d'architecture sont le plus souvent fondées sur le paradigme de programmation par échange de messages et plus particulièrement sur l'interface MPI (Message Passing Interface Forum, 1995). Les solutions de tolérance aux fautes les plus adaptées dans ce cas sont fondées sur le recouvrement arrière (Elnozahy *et al.*, 2002) : l'état de l'application est sauvegardé périodiquement sur un support de stockage stable, pour pouvoir restaurer un état intermédiaire après une défaillance, et ainsi éviter de perdre tous les calculs déjà effectués.

Pour définir une solution de tolérance aux fautes efficace dans ce contexte, plusieurs critères doivent être pris en considération. Le premier d'entre eux est la performance : l'exécution d'une application doit se terminer le plus rapidement possible en dépit des défaillances. Kogge *et al.* (2008) ont mis en évidence d'autres points cruciaux dans la perspective de la conception de machines exaflopiques, notamment la consommation énergétique et l'utilisation de la mémoire. Une solution de tolérance aux fautes adaptée doit donc être économe en espace mémoire et en terme d'énergie, le coût en terme énergétique dépendant ici principalement de la quantité de calcul à ré-exécuter après une défaillance.

Pour traiter ces enjeux, le point crucial à prendre en compte dans les protocoles de recouvrement arrière est la manière dont sont traités les messages échangés par les processus de l'application au cours de l'exécution. Dans les protocoles de sauvegarde de points de reprise coordonnés, aucun message applicatif n'est sauvegardé, où seulement les quelques messages nécessaires pour assurer un état global cohérent (Chandy *et al.*, 1985). Ainsi ce type de protocole fournit de très bonnes performances en fonctionnement normal et limite l'espace de stockage utilisé. Cependant, quand un processus subit une défaillance, tous les processus doivent redémarrer depuis le dernier point de reprise coordonné. Dans les protocoles à enregistrement de messages, tous les messages sont sauvegardés durant l'exécution normale de l'application pour pouvoir être rejoués en cas de défaillance. Ceci permet de confiner les conséquences des défaillances : avec une approche pessimiste ou causale (Alvisi *et al.*, 1998), seuls les processus fautifs doivent redémarrer après une défaillance. Mais enregistrer tous les messages peut entraîner un surcoût sur les performances des communications, et surtout implique de stocker de grandes quantités de données.

Pour tenter de combiner les avantages des protocoles de sauvegarde de points de reprise coordonnés et des protocoles à enregistrement de messages, des approches hiérarchiques ont été proposées (Vaidya, 1993). Les processus de l'application sont divisés en groupes. Un protocole de sauvegarde de points de reprise coordonnés est appliqué au sein de chaque groupe et tous les messages entre les groupes sont sauvegardés (Ho *et al.*, 2008; Yang *et al.*, 2009; Meneses *et al.*, 2010; Bouteiller *et al.*, 2011a). En regroupant les processus communiquant fréquemment entre eux, ce type de solution permet de confiner les conséquences des défaillances en enregistrant peu de messages : en cas de défaillance, seuls les processus appartenant au même groupe que le processus fautif doivent effectuer un retour arrière.

Dans cet article, nous définissons un ensemble de propriétés que nous considérons nécessaires pour un protocole de recouvrement arrière visant des applications à échange de messages s'exécutant sur des super-calculateurs de très grande taille. Nous présentons une étude détaillée des protocoles de recouvrement arrière existants, et notamment des protocoles hiérarchiques, et montrons qu'aucun d'entre eux n'a toutes les propriétés désirées. En utilisant le modèle d'exécution à émissions déterministes (Guermouche *et al.*, 2011), nous proposons un nouveau protocole hiérarchique, nommé HydEE, qui combine sauvegarde de points de reprise coordonnée et enregistrement de messages. HydEE est capable de confiner les conséquences d'une défaillance aux processus d'un seul groupe, sans sauvegarder sur support stable d'information sur les messages échangés durant l'exécution normale de l'application. Nous fournissons une évaluation de notre protocole fondée sur l'étude des schémas de communications dans les applications MPI de calcul haute performance. Nous montrons que pour la plupart des applications, enregistrer le contenu de moins de 15% des messages permet de limiter les conséquences des défaillances à moins de 15% des processus de l'application.

L'article est organisé comme suit. Le paragraphe 2 présente le contexte de nos travaux. Il définit le modèle que nous considérons, et présente les fondements des protocoles de recouvrement arrière, en se concentrant notamment sur les différents modèles d'exécution d'applications parallèles à échange de messages existants. Dans le paragraphe 3, nous étudions la problématique de la tolérance aux fautes dans les super-calculateurs pour déterminer les propriétés nécessaires pour un protocole de recouvrement arrière. Le paragraphe 4 présente un état de l'art des protocoles de recouvrement arrière hiérarchiques existants. Puis nous décrivons HydEE dans le paragraphe 5, et présentons son évaluation dans le paragraphe 6. Enfin, nous dressons les conclusions de ces travaux et proposons quelques perspectives dans le paragraphe 7.

2. Contexte

Nous commençons par définir notre système d'étude. Puis nous décrivons les principes de fonctionnement des protocoles de recouvrement arrière. Enfin, nous présentons les modèles de déterminisme des applications utilisés dans le contexte du recouvrement arrière, et décrivons leurs conséquences sur les protocoles conçus.

2.1. *Modèle*

Notre étude se place dans le cadre d'un système distribué asynchrone. Une exécution parallèle est modélisée par un ensemble fini de processus et un ensemble fini de canaux reliant chaque paire de processus. Les canaux sont fiables et délivrent les messages dans l'ordre dans lequel ils sont envoyés (FIFO). Les processus communiquent en s'échangeant des messages. Les évènements associés aux émissions et réceptions de messages sont partiellement ordonnés par la relation de précédence causale de Lamport (1978). Les fautes considérées sont des pannes franches de processus et plusieurs fautes simultanées sont possibles.

2.2. *Principes des protocoles de recouvrement arrière pour les applications à échange de messages*

Les techniques de recouvrement arrière sont fondées sur la sauvegarde de l'image des processus au cours de l'exécution normale de l'application, pour pouvoir redémarrer depuis un point intermédiaire de l'exécution après une défaillance et ainsi éviter de perdre tous les calculs déjà effectués. L'image d'un processus est appelée *point de reprise*. Les points de reprises sont sauvegardés sur un support de stockage stable (Lampson *et al.*, 1979) pour pouvoir être récupérés après une défaillance. Les communications entre les processus rendant l'état de ceux-ci interdépendants, un protocole est nécessaire pour assurer qu'un *état global cohérent* puisse être trouvé après une défaillance.

2.2.1. *État global cohérent et messages orphelins*

Un état global cohérent peut être simplement défini comme un état de l'application qui aurait pu être vu lors d'une exécution sans fautes. Si un événement est inclus dans un état global cohérent, alors tous les événements précédant causalement cet évènement le sont aussi. Dans le contexte d'une application composée de processus communiquant par échange de messages, la notion d'état global cohérent est associée à celle de *message orphelin*. Un message est orphelin si l'évènement associé à la réception d'un message fait partie de l'état global alors que l'évènement associé à son émission n'en fait pas partie. Un état global cohérent est alors un état sans message orphelin.

2.2.2. *Protocoles de sauvegarde de points de reprise*

La première famille de protocoles de recouvrement arrière sont les protocoles de sauvegarde de points de reprise non coordonnés (Bhargava *et al.*, 1988). Les processus sauvegardent des points de reprise de manière indépendante lors de l'exécution de l'application. Quand une faute se produit, le protocole tente de trouver un état global cohérent à partir de l'ensemble des points de reprise de processus indépendants qui ont été sauvegardés lors de l'exécution. Cependant, rien ne garantit qu'un état global cohérent autre que l'état initial de l'application puisse être trouvé. Le retour arrière

d'un processus peut alors entraîner une cascade de retours arrière sur l'ensemble des processus : c'est *l'effet domino* (Randell, 1975).

Pour éviter cet effet domino, les protocoles de sauvegarde de points de reprise coordonnés assurent que l'état global sauvegardé soit cohérent. La coordination des processus au moment de la sauvegarde peut être bloquante (Tamir *et al.*, 1984), non bloquante (Chandy *et al.*, 1985) ou temporelle (Neves *et al.*, 1996). En cas de défaillance, tous les processus redémarrent depuis le dernier état global cohérent sauvegardé. En prenant en compte les dépendances causales entre les processus, il est théoriquement possible de ne faire revenir en arrière que les processus dont l'état dépend de celui du processus fautif (Koo *et al.*, 1986).

Les protocoles de sauvegarde de points de reprise causaux assurent la sauvegarde d'un état global cohérent sans synchronisation explicite des processus (Briatico *et al.*, 1984). Des informations sont attachées sur chaque message, pour que le récepteur puisse déterminer si un nouveau message arrivant est susceptible de devenir orphelin, et ainsi puisse prendre un point de reprise en conséquence. La sauvegarde des points de reprise est alors dirigée par les schémas de communications au sein des applications.

2.2.3. *Protocoles à enregistrement de messages*

Les protocoles de sauvegarde de points de reprise considèrent un modèle d'exécution des applications non déterministe. En considérant un modèle d'exécution déterministe par morceaux (voir paragraphe 2.3), il est possible de définir une autre famille de protocoles de recouvrement arrière : les protocoles à enregistrement de messages, aussi appelés protocoles de journalisation.

Une application est déterministe par morceaux si il est possible de capturer, sauvegarder et rejouer après une défaillance, tous les évènements non déterministes se produisant au cours de l'exécution de l'application. Dans les applications de calcul haute performance à échange de messages, ces événements se réduisent la plupart du temps à la réception des messages. Ainsi, après une défaillance, il est possible de rétablir le processus dans l'état précédant la faute, en rejouant la même séquence de messages que lors de la première exécution.

Dans les protocoles à enregistrement de messages, le contenu de tous les messages ainsi que les informations nécessaires pour les rejouer dans le même ordre, c'est-à-dire les déterminants des messages (Alvisi *et al.*, 1998), sont sauvegardés. Ces protocoles peuvent être combinés avec des points de reprise non coordonnés, sans risque d'effet domino. Les points de reprise servent à limiter l'étendu du retour arrière en cas de défaillances, et permettent de réduire le taille des journaux. L'enregistrement de message fondé sur l'émetteur (Johnson *et al.*, 1987) permet de sauvegarder le contenu des messages dans la mémoire de leurs émetteurs. Ainsi, seuls les déterminants des messages sont sauvegardés sur support stable.

Les protocoles à enregistrement de messages se divisent en trois types : optimistes, pessimistes et causaux. Ils se différencient par la manière de traiter les déterminants (Alvisi *et al.*, 1998). Les protocoles optimistes sauvegardent les détermi-

nants de manière asynchrone sur support stable. En cas de défaillance, des déterminants peuvent être perdus, entraînant la création de messages orphelins. Tous les processus dépendant de messages orphelins doivent alors effectuer un retour arrière. Pour accélérer la recherche des processus dépendants de messages orphelins après une défaillance, les dépendances entre les processus de l'application sont en général tracées durant l'exécution normale au moyen de vecteurs de dépendances (Strom *et al.*, 1985; Sistla *et al.*, 1989) ou de vecteurs d'horloge tolérants aux fautes (Damani *et al.*, 1996; Smith *et al.*, 1996) attachés à chaque message. Dans les protocoles pessimistes, les déterminants sont sauvegardés de manière synchrone sur support stable (Borg *et al.*, 1983). Ainsi, une défaillance ne peut pas créer de messages orphelins : seuls les processus fautifs reviennent en arrière. Les protocoles causaux assurent aussi l'absence de messages orphelins après une défaillance. Pour cela, ils attachent sur chaque message, toutes les informations nécessaires au rejeu du message (Elnozahy *et al.*, 1992), c'est-à-dire les déterminants des messages dont il dépend.

Nous regroupons l'ensemble des protocoles décrits dans ce paragraphe sous le nom de protocoles de recouvrement arrière "à plat", car dans toutes ces solutions le même protocole est exécuté sur tous les canaux de communication.

2.3. *Déterminisme des applications à échange de messages*

Comme nous l'avons vu précédemment, le modèle d'exécution des applications considéré a des conséquences sur le type de protocoles qui peut être conçu, et donc sur ses caractéristiques. Dans toutes les familles de protocoles que nous avons présentés jusqu'ici, l'exécution des applications est considérée comme non déterministe. Plus précisément, les protocoles de sauvegarde de points de reprise ne font aucune supposition sur le déterminisme des applications :

Définition 1 (Modèle d'exécution non déterministe)**.** *L'exécution d'une application est considérée comme non déterministe par les protocoles de recouvrement arrière, si elle peut inclure des évènements non déterminismes dont il est impossible de garantir le rejeu à l'identique après une défaillance.*

Les protocoles à enregistrement de messages ne font pas non plus de suppositions sur le déterminisme de l'exécution des applications. Cependant, ils considèrent un modèle d'exécution un peu différent. Ils supposent que l'exécution des applications peut être modélisée comme une exécution déterministe par morceaux :

Définition 2 (Modèle d'exécution déterministe par morceaux)**.** *L'exécution d'une application est déterministe par morceaux si tous les évènements non déterministes se produisant durant l'exécution peuvent être enregistrés pour être rejoués à l'identique après une défaillance.*

Pour améliorer les protocoles de recouvrement arrière dans le cadre de l'exécution d'applications MPI de calcul haute performance, des travaux récents tentent de trouver un modèle d'exécution plus proche du comportement réel de ces applications.

Bouteiller *et al.* (2010) ont proposé de raffiner le modèle d'exécution déterministe par morceaux pour prendre en compte la sémantique de l'interface MPI. Dans les travaux considérant une exécution déterministe par morceaux précédemment cités, un évènement non déterministe est associé à la réception de chaque message. Cependant la plupart des primitives de communication MPI ont un comportement déterministe. Seul l'émission d'une requête de réception de message dans laquelle l'émetteur du message n'est pas précisée (utilisation de *MPI_ANY_SOURCE*), ou la complétion asynchrone d'une requête de réception (par exemple, utilisation de *MPI_Test()* ou *MPI_Wait_Any()*) impliquent un comportement non déterministe. Ainsi, en utilisant ce modèle, les protocoles à enregistrement de messages ont beaucoup moins d'évènements non déterministes à enregistrer que dans le modèle classique, ce qui permet d'améliorer leurs performances (Bouteiller *et al.*, 2009).

Cappello *et al.* (2010) ont proposé un nouveau modèle de déterminisme, appelé *déterminisme des émissions*, pour mieux représenter le fonctionnement des applications parallèles de calcul scientifique.

Définition 3 (Modèle d'exécution à émissions déterministes)**.** *L'exécution d'une application est à émissions déterministes, si pour un ensemble de paramètres d'entrée donné, la séquence des émissions de messages de chaque processus est la même pour toute exécution correcte.*

L'analyse d'un ensemble représentatif d'applications MPI de calcul scientifique a montré qu'une grande majorité d'entre elles ont une exécution à émissions déterministes (Cappello *et al.*, 2010). En s'appuyant sur cette propriété, il a été montré qu'il est possible de concevoir un protocole de sauvegarde de points de reprise non coordonné qui ne souffre pas de l'effet domino, et ce en ne sauvegardant le contenu que de quelques messages dans la mémoire de leur émetteur (Guermouche *et al.*, 2011).

3. Tolérance aux fautes dans les supers-calculateurs

Les machines parallèles de très grande taille ont besoin de solutions de tolérance aux fautes adaptées. Les travaux analysant ces besoins dans le cadre d'études sur les futures machines exaflopiques ont mis en évidence plusieurs point cruciaux (Kogge *et al.*, 2008; Elnozahy *et al.*, 2008; Cappello, 2009). Dans ce paragraphe, nous nous appuyons notamment sur ces études pour déterminer un ensemble de propriétés souhaitées pour un protocole de recouvrement arrière dans ce contexte. Nous analysons ensuite les familles de protocoles présentées dans le paragraphe 2 et montrons qu'ils ne peuvent fournir toutes les propriétés désirées.

3.1. *Propriétés souhaitées*

Dans la perspective de l'exécution d'applications parallèles sur des machines exaflopiques, trois enjeux majeurs sont associés à la tolérance aux fautes : (i) les performances de l'application ; (ii) la gestion de l'espace de stockage et plus particulièrement de la mémoire ; (iii) la consommation d'énergie.

3.1.1. *Performances en fonctionnement normal*

Nous considérons trois sources principales de dégradation des performances d'une application en fonctionnement normal pour un protocole de recouvrement arrière.

Congestion à la sauvegarde des points de reprise : Les protocoles de recouvrement arrière où la sauvegarde des points de reprise des différents processus de l'application se fait au même moment peuvent entraîner une dégradation des performances de l'application, due à la congestion du serveur de stockage. Oldfield *et al.* (2007) ont montré à partir des caractéristiques des supers-calculateurs actuels, et en considérant l'évolution vers les machines exaflopiques, que près de 50% du temps d'exécution pourrait être dédié à la sauvegarde des points de reprise. Il est donc important de pouvoir ordonnancer la sauvegarde des points de reprise des processus pour éviter cette congestion.

Données attachées sur les messages applicatifs : Certains protocoles ont besoin d'attacher des données sur les messages applicatifs. Il a été montré que quand la taille de ces données devient importante, ceci peut grandement pénaliser les performances des applications (Ropars *et al.*, 2009). Cependant attaché une petite quantité de données sur chaque message, par exemple un ou deux entiers, a un faible impact sur les performances (Guermouche *et al.*, 2011).

Synchronisations avec un serveur de stockage stable : Au-delà de la sauvegarde des points de reprise, les protocoles à enregistrement de messages doivent sauvegarder des données supplémentaires sur support stable durant l'exécution de l'application. Ces sauvegardes peuvent aussi dégrader les performances de l'application.

3.1.2. *Stockage de données*

Les données à sauvegarder par les protocoles de recouvrement arrière sont principalement les points de reprise des processus et le contenu des messages. Les deux principaux problèmes que nous associons à la sauvegarde de ces données sont l'utilisation de l'espace mémoire des noeuds d'exécution pour la sauvegarde des données à court terme, et de l'espace de stockage pour la sauvegarde des données de manière stable.

Sauvegardes en mémoire : Les protocoles de recouvrement arrière peuvent avoir un certains nombre de données à stocker qui n'ont pas besoin d'être sauvegardées

de manière stable. Dans ce cas, pour des raisons de performances, ces données peuvent être sauvegardées dans la mémoire des nœuds d'exécution au lieu de les écrire sur support stable. Cependant, plusieurs rapports concernant les architectures exaflopiques mentionnent une réduction de la mémoire par cœur par rapport à la situation actuelle (Kogge *et al.*, 2008). L'espace mémoire doit donc être économisé.

Efficacité du ramasse-miettes : Pour limiter l'espace de stockage utilisé, il est important de pouvoir supprimer les données obsolètes. Une donnée sauvegardée par un protocole de recouvrement arrière devient obsolète, quand il est certain qu'aucun retour arrière ne peut ramener l'application dans un état où cette donnée serait nécessaire. Dans cet article, nous appelons *ligne de recouvrement minimale*, l'état global au-delà duquel l'application ne reviendra pas, quelles que soient les défaillances. Les données concernant un état précédent cette ligne de recouvrement minimale peuvent être supprimées. Il est donc important qu'un protocole de recouvrement arrière garantisse la progression de la ligne de recouvrement minimale.

3.1.3. *Consommation d'énergie*

Dans notre étude, nous considérons que la principale source de consommation supplémentaire d'énergie liée à la tolérance aux fautes sont les retours arrière. D'autres points peuvent avoir un impact sur la consommation d'énergie, comme les surcoûts sur les performances qui impliquent un temps d'exécution plus long, ou le coût de la sauvegarde des données. Mais les surcoûts liés aux performances sont déjà pris en compte dans les critères définis dans le paragraphe 3.1.1. et il n'existe à notre connaissance pas de modèle dans la littérature pour évaluer le second.

Confinement des défaillances : Les retours arrières peuvent être limités dans le temps et dans l'espace. Limiter les retours arrière dans le temps implique d'augmenter la fréquence de sauvegarde des points de reprise des processus. Des travaux se sont intéressés au calcul d'un intervalle de sauvegarde de points de reprise optimal (Daly, 2006). Cependant, ces travaux ne sont applicables qu'aux protocoles de sauvegarde de points de reprise coordonnés, car ils considèrent chaque point de reprise comme la sauvegarde de l'état de toute l'application. C'est pourquoi nous nous concentrons sur la limitation des retours arrière dans l'espace. Pour cela, un protocole de recouvrement arrière doit être capable de confiner les conséquences d'une défaillance.

3.2. *Évaluation des protocoles de recouvrement arrière "à plat"*

Le tableau 1 présente les propriétés des protocoles de recouvrement arrière décrits dans le paragraphe 2. Au lieu de considérer les propriétés théoriques des protocoles, nous avons étudié les résultats observés lors de l'utilisation de ces protocoles pour l'exécution d'applications de calcul haute performance.

Le principal problème des protocoles de sauvegarde de points de reprise est qu'ils ne permettent pas de confiner les défaillances. En utilisant la solution proposée par Koo *et al.* (1986), il est théoriquement possible de ne faire redémarrer qu'un sous ensemble des processus après une défaillance en utilisant un protocole de sauvegarde de points de reprise coordonnés : seuls les processus dépendant causalement d'un processus fautif doivent redémarrer. Cependant les expériences présentées par Guermouche *et al.* (2011) montrent que lors de l'exécution des *NAS Parallel Benchmarks*, tous les processus de l'application deviennent causalement dépendants très rapidement au cours de l'exécution. Ceci implique que les protocoles n'utilisant pas d'enregistrement de messages ne peuvent pas confiner les défaillances.

Les protocoles de sauvegarde de points de reprise induits par les communications ont été conçus pour assurer la présence d'un état global cohérent sans synchronisation explicite des processus. Cependant, l'évaluation de ce type de protocoles avec des applications de calcul haute performance montre que le nombre de points de reprise forcés pour assurer la présence d'un état global cohérent est bien supérieur au nombre de points de reprise pris indépendamment (Alvisi *et al.*, 1999). Ils ne permettent donc pas d'ordonnancer la sauvegarde de points de reprise.

Il est important de noter que des solutions existent pour limiter la congestion du serveur de stockage à la sauvegarde des points de reprise dans un protocole coordonné. Des solutions de sauvegarde de points de reprise à plusieurs niveaux (Moody *et al.*, 2010) sauvegardent certains points de reprise sur les supports de stockage locaux des nœuds d'exécution (mémoire vive, disque Flash), pour limiter la surcharge du serveur de stockage. Il s'agit alors d'un compromis car ces solutions de stockage tolèrent moins de fautes.

Les protocoles à enregistrement de messages permettent de confiner les défaillances. Avec un protocole à enregistrement de messages pessimiste ou causal, seuls les processus fautifs effectuent un retour arrière après une défaillance. De plus, ils peuvent être combinés avec un protocole de sauvegarde de points de reprise non coordonnés sans risque d'effet domino, et permettent donc d'ordonnancer la sauvegarde des points de reprise. Le principal problème vient alors du coût de la sauvegarde des messages. Pour rendre ce coût acceptable, il est indispensable de sauvegarder le contenu des messages dans la mémoire de leur émetteur, ce qui est très coûteux en espace mémoire. De plus, la sauvegarde des déterminants sur support stable entraîne souvent une dégradation, même en essayant de mettre en œuvre cette sauvegarde de manière distribuée pour améliorer ce point (Ropars *et al.*, 2011b).

L'utilisation du modèle d'exécution proposé par Bouteiller *et al.* (2010) permet de limiter le nombre de déterminants à sauvegarder au cours de l'exécution de l'application et donc l'impact sur les performances (Bouteiller *et al.*, 2009). Cependant, il n'est pas certain que cette solution permette de réduire suffisamment le nombre de déterminants à large échelle et pour toutes les applications.

Pour se rapprocher d'un protocole ayant toutes les propriétés requises pour l'exécution d'applications parallèles à large échelle, nous avons proposé un protocole de

Protocole	Ordonnancement des points de reprise	Données attachées sur les messages	Synchronisation avec stockage stable	Sauvegardes en mémoire	Progression de la ligne de recouvrement minimale	Confinement des défaillances
Coordonné	Non	**Aucune**	**Aucune**	**Aucune**	**Oui**	Non
Non coordonné	**Possible**	Vecteur de dépendance	**Aucune**	**Aucune**	Non	Non
Induit par les communications	Non	Entier ou vecteur d'entiers	**Aucune**	**Aucune**	**Oui**	Non
Pessimiste	**Possible**	**Aucune**	Sauvegarde synchrone des déterminants	Contenu de tous les messages	**Oui**	**Oui**
Optimiste	**Possible**	Vecteur de dépendance	Sauvegarde asynchrone des déterminants	Contenu de tous les messages	**Oui**	Partiel
Causal	**Possible**	Déterminants non stable	Sauvegarde asynchrone des déterminants	Contenu de tous les messages	**Oui**	**Oui**
Guermouche *et al.* (2011)	**Possible**	**Entiers**	**Aucune**	**Contenu d'une partie des messages**	**Oui**	Partiel

Tableau 1. *Évaluation des protocoles de recouvrement arrière "à plat"*

sauvegarde de points de reprise non coordonnés (Guermouche *et al.*, 2011), exploitant le modèle d'exécution à émissions déterministes (définition 3). Comme c'est un protocole de sauvegarde de points de reprise non coordonnés, il permet d'ordonnancer la sauvegarde des points de reprise. Les points de reprise de chaque processus sont identifiés par leur numéro d'époque, incrémenté à chaque nouveau point de reprise. Ce protocole assure que les points de reprise de processus avec le même numéro d'époque font partie d'un état global cohérent. Pour rendre cet état global cohérent, et ainsi éviter l'effet domino, le contenu des messages dont l'époque d'émission est inférieure à l'époque de réception est enregistré dans la mémoire de l'émetteur. Ainsi seul le contenu d'un sous ensemble des messages est sauvegardé. De plus, ce protocole utilise les propriétés du déterminisme des émissions pour ne pas enregistrer les déterminants des messages, évitant ainsi toute synchronisation avec un support de stockage stable, si ce n'est pour la sauvegarde des points de reprise. Le problème principal de ce protocole est que, par défaut, il ne permet pas de confiner les défaillances. Nous avons proposé une utilisation de ce protocole fondée sur le découpage des processus de l'application en groupes. En attribuant à chaque groupe une époque initiale différente, cette solution force l'enregistrement de tous les messages allant d'un groupe vers un groupe dans une époque supérieure. Ainsi en cas de défaillance d'un processus, seuls les processus du même groupe, et des groupes ayant une époque supérieure, doivent effectuer un retour arrière. Cette solution permet de confiner partiellement les défaillances en n'enregistrant qu'une sous partie des messages. Cependant si le pro-

cessus fautif appartient au groupe avec la plus petite époque, tous les processus de l'application doivent effectuer un retour arrière.

4. État de l'art des protocoles de recouvrement arrières hiérarchiques

Vaidya (1993) a défini la notion de protocole de recouvrement arrière hybride. Contrairement aux protocoles "à plat" décrits dans les paragraphes précédent, deux protocoles différents sont utilisés selon les canaux de communications. Des groupes sont définis au sein des processus de l'application. Un protocole de recouvrement arrière *local* est utilisé au sein de chaque groupe, et un protocole de recouvrement arrière *global* est appliqué entre les groupes. Dans cet article, nous choisissons le terme de protocole *hiérarchique* plutôt que protocole hybride pour mieux exprimer l'idée de combinaison d'un protocole local et d'un protocole global. L'objectif des protocoles hiérarchiques est de tenter de combiner les avantages de deux protocoles "à plat".

4.1. *Choix des groupes de processus*

La manière de regrouper les processus est indépendante du choix des protocoles utilisés. Cependant, dans la plupart des cas, les groupes sont choisis de manière à ce que les processus communiquant fréquemment entre eux soient dans le même groupe. Certains regroupements sont fondés sur l'aspect hiérarchique des architectures visées : dans les architectures de type fédération de grappes de calcul, tous les processus s'exécutant sur la même grappe peuvent appartenir au même groupe (Monnet *et al.*, 2004; Gupta *et al.*, 2011; Ropars *et al.*, 2008). Il est alors supposé que les communications au sein des grappes de calcul sont plus nombreuses que les communications entre processus appartement à différentes grappes. D'autres solutions analysent directement les schémas de communications des applications pour former les groupes (Ho *et al.*, 2008). Enfin, une solution consiste à regrouper les processus ayant une forte probabilité de subir une défaillance au même moment, typiquement les processus s'exécutant sur les cœurs d'un même nœud (Bouteiller *et al.*, 2011b).

Nous avons proposé une solution pour choisir automatiquement les groupes de processus pour une application, à partir du graphe de communication de cette application (Ropars *et al.*, 2011a). Cette solution prend en paramètre une fonction de coût qui lui permet de trouver un découpage adapté aux caractéristiques du protocole hiérarchique visé. Nous donnons plus de détails sur cette solution dans le paragraphe 6.

4.2. *Description des protocoles de recouvrement arrière hiérarchiques existants*

Nous classons les protocoles de recouvrement arrière hiérarchiques existant selon le type de protocole utilisé au niveau local et au niveau global. Certains protocoles

hiérarchiques combinent le même type de protocole aux deux niveaux (Gao *et al.*, 2007; Ropars *et al.*, 2008; Bhatia *et al.*, 2003). Nous ignorons ces protocoles dans notre étude car ce type de solution ne change pas fondamentalement les propriétés des protocoles, mais améliore juste le passage à l'échelle.

Dans les protocoles hiérarchiques existants, le protocole utilisé au niveau local est toujours un protocole de sauvegarde de points de reprise coordonnés. Comme les groupes rassemblent les processus communiquant fréquemment entre eux et/ou les processus ayant une forte probabilité de subir une défaillance au même moment, un protocole coordonné est le plus adapté. Comme il est inutile d'essayer de confiner les défaillances, c'est le type de protocole qui offre les meilleures performances en fonctionnement normal tout en minimisant l'espace de stockage utilisé. Nous présentons, dans un premier temps, les solutions utilisant un autre protocole de sauvegarde de points de reprise au niveau global, puis les solutions utilisant un protocole à enregistrement de messages.

4.2.1. *Sauvegarde de points de reprise au niveau global*

Monnet *et al.* (2004) et Gupta *et al.* (2011) ont proposé d'utiliser un protocole de sauvegarde de points de reprise induit par les communications au niveau global. Ces deux protocoles visent des systèmes de type fédération de grappes de calcul et se différencient seulement par la manière de calculer la ligne de recouvrement après une défaillance. Ces deux protocoles souffrent des problèmes que nous avons énoncés dans le paragraphe 3.2 pour les protocoles de sauvegarde de points de reprise induits par les communications. Ils ne permettent pas en pratique d'ordonnancer la sauvegarde des points de reprise et ils ne permettent pas de confiner les défaillances.

4.2.2. *Enregistrement de messages au niveau global*

La plupart des protocoles présentés dans ce paragraphe considèrent le modèle d'exécution déterministe par morceaux (définition 2). Bouteiller *et al.* (2011b) ont montré que, dans ce modèle, pour être capable de redémarrer une application en utilisant un protocole hiérarchique combinant sauvegarde de points de reprise au niveau local et protocole à enregistrement de messages au niveau global, il faut sauvegarder les déterminants de tous les messages de l'application, y compris ceux échangés au sein des groupes.

Vaidya (1993) fut le premier à combiner protocole de sauvegarde de points de reprise coordonnés au niveau local et enregistrement de messages au niveau global. Il proposa d'utiliser un protocole à enregistrement de messages optimiste. Cependant, ce type de protocole ne permet pas de confiner les défaillances.

D'autres protocoles utilisent un protocole pessimiste entre les groupes pour assurer que la défaillance d'un processus ne fasse revenir en arrière que les processus du groupe auquel il appartient (Yang *et al.*, 2009; Meneses *et al.*, 2010; Bouteiller *et al.*, 2011b). Yang *et al.* (2009) proposent une solution inspirée des protocoles à enregistrement de messages causaux pour gérer les déterminants des messages intra-

groupe. Les déterminants qui ne sont pas encore sauvegardés sur support stable, et dont un message dépend, sont attachés sur ce message. Quand un message doit être envoyé vers un autre groupe, tous les déterminants des messages dont il dépend sont préalablement sauvegardés sur support stable. L'alternative consiste à gérer ces déterminants de manière pessimiste (Meneses *et al.*, 2010; Bouteiller *et al.*, 2011b), ce qui implique des synchronisations plus fréquentes avec le serveur de stockage, mais évite d'attacher des données sur les messages.

Enfin d'autres protocoles qui combinent sauvegarde de points de reprise coordonnés et enregistrement de messages, ne sauvegardent pas les déterminants des messages échangés au niveau local (Ho *et al.*, 2008; Gupta *et al.*, 2008). Ces protocoles ne sont valides que pour des applications complètement déterministes.

4.3. *Évaluation des protocoles de recouvrement arrière hiérarchiques*

Nous utilisons les critères présentés dans le paragraphe 3.1.1 pour évaluer les protocoles hiérarchiques existants. Le tableau 2 présente cette évaluation. Dans cette évaluation, nous ne considérons pas les protocoles ne fonctionnant que pour des applications complètement déterministes. Nous ne traitons pas non plus les protocoles présentés par Bouteiller *et al.* (2011b) et par Gupta *et al.* (2011) car, du point de vue de notre étude, ils sont identiques aux protocoles présentés par Meneses *et al.* (2010) et Monnet *et al.* (2004) respectivement.

Protocole	Ordonnancement des points de reprise	Données attachées sur les messages	Synchronisation avec stockage stable	Sauvegardes en mémoire	Progression de la ligne de recouvrement minimale	Confinement des défaillances
Monnet *et al.* (2004)	Non	**Entiers**	**Aucune**	**Aucune**	**Oui**	Non
Vaidya (1993)	**Possible**	**Aucune**	Sauvegarde asynchrone des déterminants	**Messages inter groupes**	**Oui**	Partiel
Yang *et al.* (2009)	**Possible**	Déterminants non stable	Sauvegarde asynchrone des déterminants	**Messages inter groupes**	**Oui**	**Oui**
Meneses *et al.* (2010)	**Possible**	**Aucune**	Sauvegarde synchrone des déterminants	**Messages inter groupes**	**Oui**	**Oui**
HydEE	**Possible**	**Entiers**	**Aucune**	**Messages inter groupes**	**Oui**	**Oui**

Tableau 2. *Évaluation des protocoles de recouvrement arrière hiérarchiques*

Cette évaluation montre qu'aucun protocole existant n'arrive à fournir toutes les propriétés souhaitées dans le cadre de l'exécution d'applications parallèles sur des super-calculateurs de très grande taille. L'utilisation d'un protocole de sauvegarde

de points de reprise induit par les communications au niveau global ne permet pas d'ordonnancer les points de reprise ni de confiner les défaillances. L'utilisation d'un protocole à enregistrement messages au niveau global est une solution plus attractive. Tout d'abord, cette solution permet l'ordonnancement des points de reprise, c'est-à-dire que la sauvegarde des points de reprise coordonnés de chaque groupe peut être faite de manière indépendante. Ensuite, elle assure le confinement des conséquences d'une défaillance au groupe où cette défaillance s'est produite, si le protocole à enregistrement de messages choisi n'est pas un protocole optimiste.

Le principal point négatif des protocoles proposés par (Yang *et al.*, 2009) et (Meneses *et al.*, 2010) est lié à la gestion des déterminants des messages internes aux groupes, qui peut avoir un impact sur les performances des applications en fonctionnement normal. Il est aussi important de noter que pour ces protocoles, l'efficacité du confinement des défaillances est proportionnel à la taille des groupes. Cependant réduire la taille des groupes a pour conséquence d'augmenter le nombre de messages inter-groupe, et donc l'espace mémoire utilisé pour sauvegarder le contenu de ces messages. Un compromis doit être trouvé à ce niveau.

Dans le paragraphe 5, nous décrivons HydEE, un protocole de recouvrement arrière hiérarchique combinant sauvegarde de points de reprise coordonnés au niveau local, et enregistrement de messages au niveau global. HydEE est fondé sur le modèle d'exécution à émissions déterministe (définition 3), pour assurer le bon redémarrage des applications après une défaillance, tout en ne sauvegardant aucun déterminant durant l'exécution normale de l'application. Dans le paragraphe 6, nous présentons une évaluation de HydEE montrant sur un ensemble représentatif d'applications le compromis qui peut être trouvé entre taille des groupes et quantité de messages enregistrés

5. Un protocole de recouvrement arrière hiérarchique pour applications à échange de messages avec émissions déterministes

Nous proposons un nouveau protocole hiérarchique, nommé HydEE, fondé sur la combinaison d'un protocole de sauvegarde de points de reprise coordonnés au niveau local et d'un protocole à enregistrement de messages au niveau global. Ce protocole considère le modèle d'exécution à émissions déterministes (définition 3).

5.1. *HydEE en fonctionnement normal*

L'algorithme 1 décrit le fonctionnement de HydEE lors d'une exécution normale, c'est-à-dire quand il n'y a pas de fautes. Dans ce protocole, le contenu des messages échangés entre les groupes est enregistré au niveau de l'émetteur. Quand un processus envoie un message à un processus appartenant à un autre groupe, il enregistre le contenu du message dans sa mémoire. À chaque processus est associé une date qui est incrémentée à chaque émission ou réception de message. La date d'émission des messages inter-groupe est également enregistrée afin d'identifier les messages à ren-

voyer en cas de défaillance. Au sein d'un groupe, le protocole utilisé est un protocole de sauvegarde de points de reprise coordonnés. Notons que le choix du protocole coordonné utilisé est indépendant de notre protocole. C'est pourquoi, dans un souci de simplicité, nous n'incluons pas la sauvegarde des points de reprise coordonnés dans le pseudo-code décrivant notre protocole. Le calcul d'un état global cohérent n'a pas besoin de tenir compte des messages inter-groupe car ils sont sauvegardés dans la mémoire de leur émetteur. Au moment de la sauvegarde d'un point de reprise sur support stable, sont aussi sauvegardés les messages ayant été copiés en mémoire.

Algorithm 1 Algorithme des processus de l'application en fonctionnement normal

Variables Locales:

P_i {Identifiant du processus i}
$Groupe_i$ {Groupe auquel le processus i appartient}
$date_i \leftarrow 0$ {Date du processus i}
$phase_i \leftarrow 0$ {Phase du processus i}
$enregistres_i \leftarrow \emptyset$ {Ensemble des messages enregistrés par le processus i}

Envoyer le message msg **au processus** P_j
  $date_i \leftarrow date_i + 1$
  si $Groupe_i \neq Groupe_j$ **alors**
    Enregistrer msg et $date_i$ dans $enregistres_i$
  Envoyer(msg, $date_i$, $phase_i$, $Groupe_i$) au processus P_j

Délivrer le message (msg**,** $date_{msg}$**,** $phase_{msg}$**,** $Groupe_{msg}$**) du processus** P_j
  $date_i \leftarrow date_i + 1$
  si $Groupe_i \neq Groupe_{msg}$ **alors**
    $phase_i \leftarrow maximum(phase_i, phase_{msg} + 1)$
  sinon
    $phase_i \leftarrow maximum(phase_i, phase_{msg})$
  Délivrer msg à l'application

5.2. *HydEE en redémarrage*

Lorsqu'un processus subit une défaillance, tous les processus du même groupe font un retour arrière. Les processus des autres groupes ne font pas de retour arrière car : 1) les messages destinés aux processus du groupe fautif ont été enregistrés et 2) le déterminisme des émissions assure que les messages devenus orphelins lors du retour arrière restent valides. Cependant, le protocole doit garantir que l'ordre causal des messages soit respecté lors du rejeu. En d'autres termes, si l'émission d'un message m' dépend de la réception d'un message m, il faut assurer que durant le redémarrage, m' n'est pas renvoyé avant que m ne soit reçu. Afin d'illustrer le problème, la figure 1 présente une exécution avec 8 processus divisés en 3 groupes. La figure 2 est la représentation temporelle de la figure 1. Sur ces figures, les indices des messages

représentent leur ordre causal. Les messages m_1, m_3 et m_7 sont des messages inter-groupe. Dans une exécution sans faute, les messages m_3 et m_7 ne peuvent être émis avant la réception de m_1 et m_3 respectivement. Après la défaillance d'un processus du $Groupe_2$, tous les processus de ce groupe effectuent un retour arrière vers leur dernier point de reprise. Le message m_3 devient donc un message orphelin. Cependant, comme le déterminisme des émissions assure l'existence du même message m_3 dans toute exécution correcte, les processus du $Groupe_3$ ne font pas de retour arrière pour recevoir m_3 à nouveau. Le message enregistré m_7 pourrait donc être renvoyé avant m_1 et entraînant la réception de m_8 avant m_2. Cependant, étant donné que m_8 dépend de m_2, l'exécution ne serait plus cohérente.

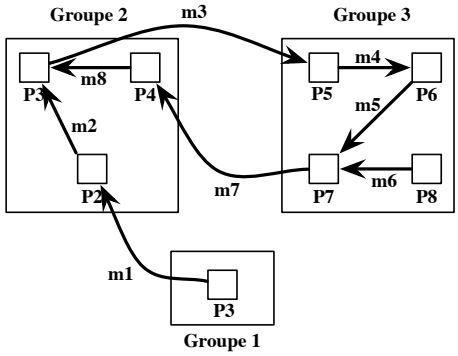

Figure 1. *Exemple de scenario d'exécution de HydEE*

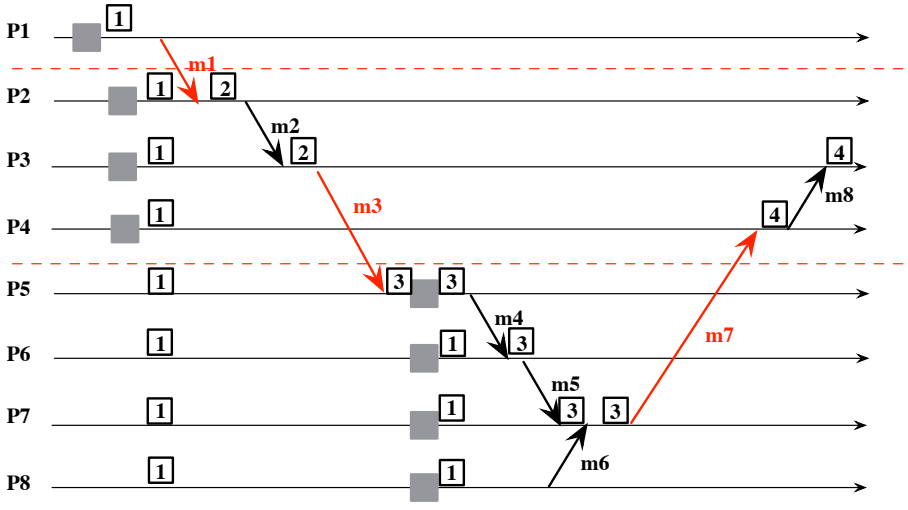

Figure 2. *Représentation temporelle du scénario d'exécution*

La différence entre une exécution en fonctionnement normal et en redémarrage réside dans l'existence des messages orphelins : les processus du $Groupe_3$ n'ont pas besoin de recevoir m_3 pour envoyer leurs messages enregistrés. C'est pourquoi il est nécessaire d'avoir un moyen de savoir quand un message dépend d'un message orphelin.

Afin de garantir que la causalité est respectée, nous adaptons la technique que nous avons proposé dans (Guermouche *et al.*, 2011) : utiliser des numéros de phase pour indiquer qu'un message dépend d'un potentiel orphelin. Un message m' dépendant d'un message m qui vient d'un autre groupe doit avoir un numéro de phase supérieur à celui de m. Durant le redémarrage, m' ne sera pas renvoyé tant que tous les messages orphelins à une phase inférieure à la sienne ne sont pas rejoués.

Les phases sont mises en œuvre de cette manière : chaque processus se voit attribuer un numéro de phase qui est attaché à chaque message qu'il envoie. La phase du processus est mise à jour chaque fois qu'il reçoit un message, comme suit :

– si le message vient du même groupe et qu'il a une phase supérieure, le processus prend la phase du message.

– si la phase vient d'un autre groupe, il prend le maximum entre la phase du message incrémentée de 1 et sa propre phase.

Le fonctionnement des phases est illustré sur la figure 2. Les numéros encadrés sont les numéros de phase. Après la défaillance des processus du $Groupe_2$, le message enregistré m_1 est le seul qui peut être envoyé étant donné qu'il est le seul message dont la phase est inférieure à celle du message orphelin m_3. Une fois que le message m_3 est rejoué, le message m_7 peut être renvoyé. L'ordre causal est ainsi garanti.

L'algorithme 2 décrit le fonctionnement des processus appartenant au groupe effectuant un retour arrière après une défaillance. L'algorithme 3 décrit lui le fonctionnement des processus appartenant aux autres groupes pendant cette phase de recouvrement arrière après une défaillance.

Afin d'assurer que les messages sont rejoués selon l'ordre causal après une défaillance, un processus appelé *processus de redémarrage* est utilisé. Son fonctionnement est décrit par l'algorithme 4. Le processus de redémarrage assure qu'un message inter-groupe ne peut être ré-émis tant qu'il y a des messages orphelins à une phase inférieure à la sienne. Pour ce faire, tous les processus de l'application lui envoie les phases des messages enregistrés qu'ils doivent renvoyer, ainsi que la phase des messages reçus qui deviennent orphelins. Une fois que le processus de redémarrage a reçu toutes ces données, il peut commencer la phase de notification. Afin de pouvoir renvoyer un message qui est à la phase i, il faut s'assurer que tous les messages orphelins à une phase inférieure ont bien été rejoués. Au lieu de renvoyer les messages orphelins, c'est-à-dire les messages qui ont déjà été reçus, les processus en cours de redémarrage envoient une notification au processus de redémarrage (ligne 16 de l'algorithme 2). Une fois que le processus de redémarrage reçoit toutes les notifications pour une phase, il notifie tous les processus qui ont des messages à envoyer à des

Algorithm 2 Algorithme des processus en redémarrage

Variables Locales:

1: P_i {Identifiant du processus i}
2: $Groupe_i$ {Groupe auquel le processus i appartient}
3: $date_i \leftarrow date_r$ {Date du processus i initialisée à la date du point de reprise}
4: $phase_i \leftarrow phase_r$ {Phase du processus i initialisée à la date du point de reprise}
5: $enregistres_i \leftarrow \emptyset$ {Ensemble des messages enregistrés par le processus i}
6:
7: **Au redémarrage du processus** P_i
8: Envoyer ($Redemarrage$, $date_i$, $Groupe_i$) à tous les processus P_j **tels que** $Groupe_i \neq Groupe_j$
9: Envoyer ($Phase\_Courante$, $phase_i$) au processus de redémarrage
10: Attendre ($Notification$, $phase_i$) du processus de redémarrage
11:
12: **Envoyer le message** msg **au processus** P_j
13: $date_i \leftarrow date_i + 1$
14: **si** $Groupe_i \neq Groupe_j$ **alors**
15: Enregistrer msg et $date_i$ dans $enregistres_i$
16: Envoyer ($Notification$, $phase_i$) au processus de redémarrage
17: **sinon**
18: Envoyer(msg, $date_i$, $phase_i$, $Groupe_i$) au processus P_j
19:
20: **Délivrer le message (**msg**,** $date_{msg}$**,** $phase_{msg}$**,** $Groupe_{msg}$**) du processus** P_j
21: $date_i \leftarrow date_i + 1$
22: **si** $Groupe_i \neq Groupe_{msg}$ **alors**
23: $phase_i \leftarrow maximum(phase_i, phase_{msg} + 1)$
24: **sinon**
25: $phase_i \leftarrow maximum(phase_i, phase_{msg})$
26: Délivrer msg à l'application

phases inférieures ou égales à la nouvelle phase où il n'y a plus de message orphelin (ligne 18 de l'algorithme 4). Sur la figure 1, lorsque les processus du $Groupe$ 2 reviennent en arrière à la suite d'une défaillance, les messages m_1 et m_7 sont à renvoyer. La phase de m_1 est 1 alors que celle de m_7 est 3. Étant donné qu'il n'y a pas de messages orphelins à une phase inférieure à 1, m_1 est rejoué. Cependant m_7 ne pourra être rejoué que lorsque la notification correspondant au rejeu du message m_3 aura été reçue par le processus de redémarrage. À noter enfin que même les processus n'effectuant pas de retour arrière sont bloqués dans l'attente d'une notification pour leur phase courante de la part du processus de redémarrage, ceci pour assurer qu'ils n'envoient pas de nouveaux messages qui pourraient dépendre de messages orphelins (ligne 11 de l'algorithme 3).

Algorithm 3 Algorithme des autres processus pendant le redémarrage

Variables Locales:

$enregistres_i$ {Ensemble des messages enregistrés par le processus i}
$rejouer_i \leftarrow \emptyset$ {Liste de messages à rejouer}

Sur réception de ($Redemarrage$**,**$phase_j$**,**$Groupe_j$**) du processus** P_j
  Attendre le message $Redemarrage$ de tous les processus de $Groupe_j$
  Calculer $rejouer_i$, la liste des messages de $enregistres_i$ à rejouer
  Soit $Phases\_Rejouees_i$, la liste des phases des messages dans $rejouer_i$
  Soit $Phase\_Orphelins_i$, la liste des phases des messages orphelins reçus par P_i
  Envoyer ($Phases\_Rejouees_i$, $Phase\_Orphelins_i$) au processus de redémarrage
  Envoyer ($Phase\_Courante$, $phase_i$) au processus de redémarrage
  Attendre ($Notification$, $phase_i$) du processus de redémarrage

Sur réception de ($Notification$**,** $phase_{notif}$**) du processus de redémarrage**
  pour tout $msg \in rejouer_i$ **tels que** $phase_{msg} \leq phase_{notif}$ **faire**
    Renvoyer msg
    Enlever msg de $rejouer_i$

6. Évaluation fondée sur l'analyse des schémas de communication des applications parallèles de calcul haute performance

Dans ce paragraphe, nous évaluons le comportement de notre protocole hiérarchique en étudiant les schémas de communication d'un ensemble représentatif d'application parallèles de calcul scientifique. Ce paragraphe est fondé sur une étude déjà publiée (Ropars *et al.*, 2011a), qui met en évidence la possibilité de partitionner efficacement la plupart des applications de calcul scientifique à échange de messages. Pour plus de détails, notamment sur la technique de partitionnement employée, nous invitons le lecteur à se référer à cette publication.

6.1. *Description des applications*

Asanovic *et al.* (2006) ont proposé une classification des applications de calcul haute performance selon leurs motifs de calcul et de communications. Treize motifs ont été définis, parmi lesquels sept s'appliquent aux simulations numériques : algèbre linéaire dense, algèbre linéaire creuse, méthodes spectrales, simulations N-corps, grilles structurées, grilles non structurées et MapReduce. Nous n'avons pas considérés d'applications de type MapReduce dans notre étude car les techniques de recouvrement arrière ne sont pas adaptées à ce type d'application.

Algorithm 4 Algorithme du processus de redémarrage

Variables Locales:

$Phase\_Redemarrage$ {Liste des phases des processus au moment du redémarrage}
$Phase\_Rejeux$ {Liste par phase des processus ayant un message à rejouer}
$Phase\_Orphelin$ {Nombre de messages orphelins par phase}

Sur réception de ($Phase\_Courante$**,** $phase_j$**) du processus** P_j
  Ajouter $(P_j, phase_j)$ à $Phase\_Redemarrage$

Sur réception de ($Phases\_Rejouees_i$**,** $Phase\_Orphelins_i$**) du processus** P_j
  Mettre à jour $Phase\_Rejeux$ et $Phase\_Orphelin$

Sur reception de ($Notification$**,** $phase_i$**) du processus** P_j
  $Phase\_Orphelin[phase_i] \leftarrow Phase\_Orphelin[phase_i] - 1$
  si $Phase\_Orphelin[phase_i] = 0$ **alors**
    pour tout $(P_k, phase_k) \in Phase\_Redemarrage$ **tels qu'**il n'existe plus d'orphelin dans $Phase\_Orphelin$ à une phase $phase' < phase_k$ **faire**
      Envoyer $(Notification, phase_k)$ au processus p_k
      Enlever $(P_k, phase_k)$ de $Phase\_Redemarrage$
    pour tout $(P_r, phase_r) \in Phase\_Rejeux$ **tels qu'**il n'existe plus d'orphelin dans $Phase\_Orphelin$ à une phase $phase' < phase_r$ **faire**
      Envoyer $(Notification, phase_r)$ au processus p_r
      Enlever $(P_r, phase_r)$ de $Phase\_Rejeux$

Algèbre Linéaire Dense	Algèbre Linéaire Creuse	Méthodes Spectrales	Simulations N-Corps	Grilles Structurées	Grilles Non Structurées
BT LU SP PARATEC	CG MAESTRO MILC	FT IMPACT MILC PARATEC SPECFEM3D	GTC LAMMPS MILC Nbody	CAM GTC IMPACT MAESTRO MG PARATEC SPECFEM3D	MAESTRO

Tableau 3. *Classification des applications étudiées selon les motifs de Berkeley*

Notre sélection d'applications permet de couvrir les 6 motifs précédemment cités, comme le montre le tableau 3. Elle comprend des applications appartenant à trois suites différentes : les *NAS Parallel Benchmarks* (Bailey *et al.*, 1991), les *NERSC Benchmarks* (Antypas *et al.*, 2008) et les *Sequoia Benchmarks* (asc.llnl.gov, 2009). Nous avons également étudié l'application SPECFEM3D (Komatitsch *et al.*, 2003) ainsi qu'un noyau *Nbody*.

6.2. *Méthodologie*

Afin d'obtenir un partitionnement adapté des processus pour chacune des applications, nous utilisons l'algorithme de partitionnement décrit dans (Ropars *et al.*, 2011a). Cet algorithme permet d'obtenir un partitionnement d'une application à partir du graphe décrivant les volumes de données échangées entre chaque paire de processus. Afin d'obtenir ces graphes, nous avons modifié la bibliothèque MPICH2[1] et avons exécuté l'ensemble des applications décrites sur la grappe de calcul de Rennes de la plate-forme Grid'5000. Les expériences ont été menées sur un réseau Ethernet.

L'algorithme de partitionnement est fondé sur des bissections successives. Chaque configuration obtenue est évaluée à l'aide d'une fonction de coût. Cette fonction peut être adaptée au protocole de recouvrement arrière visé. Nous utilisons une fonction de coût tentant de modéliser le comportement de HydEE. Celle-ci est de la forme :

$$\text{cout}(\Pi) = \alpha \times L + \beta \times R \quad [1]$$

où α représente le coût de l'enregistrement des messages, β le coût associé au redémarrage après une défaillance, L le ratio des données enregistrées et R le ratio des processus à redémarrer après une défaillance en considérant que les probabilités de fautes sont également reparties sur l'ensemble des processus de l'application.

Afin de fixer la valeur de β, nous avons considéré le temps perdu après une défaillance, *i.e.*, la quantité de calculs à refaire. Nous considérons un système où le temps moyen entre deux défaillances est de un jour, le temps nécessaire à la sauvegarde d'un point reprise d'une application sur support stable est de 30 minutes, tout comme le temps nécessaire pour redémarrer une application (Cappello, 2009). En utilisant la fonction proposée par Daly (2003) pour calculer un intervalle de sauvegarde de point de reprise optimal, l'étude décrite par Ropars *et al.* (2011a) montre que 12.4% du temps par jour serait dédié au recouvrement des défaillances, d'où $\beta = 12.4\%$. Nous avons fixé la valeur de α à 23%. C'est le surcoût moyen sur la latence et la bande passante introduit par l'enregistrement des messages que nous avons observés lors d'expériences sur le réseau haute performances Myrinet/MX (Guermouche *et al.*, 2011).

6.3. *Résultats*

Le tableau 4 détaille le partitionnement ainsi obtenu pour chaque application. Il présente pour chaque application, le nombre de processus sur lequel elle est exécutée, le nombre de groupes obtenus, la taille minimale et maximale des groupes, le pourcentage des processus à redémarrer en moyenne après une défaillance et le ratio de données enregistrées par rapport à la quantité totale de données échangées au cours de l'exécution de l'application.

1. http ://svn.mcs.anl.gov/repos/mpi/mpich2/trunk :r7592

Ce tableau montre que pour la plupart des applications, il est possible de créer des groupes de processus de façon à redémarrer moins de 15% des processus après une défaillance tout en enregistrant moins de 15% des messages.

L'application *FT* est la seule pour laquelle aucun partitionnement n'a pu être trouvé. Ceci s'explique par l'utilisation de primitives de communication de type *all-toall*, qui implique des communications entre tous les processus de l'application. Cependant, l'utilisation de telles primitives devrait être évitée pour des applications visant une très grande échelle.

	Nombre de processus	Nombre de groupes	Taille des groupes (Min/Max)	Pourcentage de processus à redémarrer	Quantité de données enregistrées (en Go)
NPB BT	1024	8	123/133	12.5%	201/1635 (12.3%)
NPB CG	1024	32	32/32	3.1%	910/5606 (16.2%)
NPB FT	1024	2	502/522	50%	432/864 (50%)
NPB LU	1024	16	64/64	6.25%	67/700 (9.7%)
NPB MG	1024	8	128/128	12.5%	20/107 (18.5%)
NPB SP	1024	8	123/133	12.5%	366/2989 (12.2%)
GTC	512	16	32/32	6.25%	240/3654 (6.6%)
LAMMPS	1024	8	127/129	12.5%	0.3/4 (7.6%)
MAESTRO	1024	4	252/259	25%	55/309 (17.7%)
Nbody	1024	30	31/61	3.5%	80/2733 (2.9%)
PARATEC	1024	13	64/128	8.5%	2262/23914 (9.4%)
SPECFEM3D	1215	14	76/152	7.80%	11.12%

Tableau 4. *Pourcentage des processus à redémarrer et de la quantité de données à enregistrer avec HydEE*

7. Conclusion

Dans cet article, nous avons proposé HydEE, un protocole hiérarchique pour applications à émissions déterministes. Il divise les processus de l'application en groupes pour appliquer un protocole de sauvegarde de points de reprise coordonnés au sein des groupes et un protocole à enregistrement de messages fondé sur l'émetteur pour les messages inter-groupe. Ainsi, il peut confiner les conséquences d'une défaillance à un groupe de processus. Comparé aux protocoles hiérarchiques existants, HydEE a l'avantage de ne sauvegarder aucun déterminant de messages sur support stable tout en étant capable de tolérer des fautes multiples. Enregistrer le contenu de moins de 15% des messages permet pour la plupart des applications de limiter les retours arrière à moins de 15% des processus.

L'étude de l'état de l'art des protocoles de recouvrement arrière "à plat" et hiérarchiques suggère que HydEE est le mieux adapté pour répondre aux besoins en tolérance aux fautes des futures architectures exaflopiques. C'est le protocole qui offre le meilleur compromis entre performances en fonctionnement normal, usage limitée de l'espace mémoire, et consommation d'énergie liée aux retours arrières.

Pour pouvoir analyser expérimentalement les performances de HydEE, sa mise en œuvre dans une bibliothèque MPICH2 est en cours. En particulier, nous voudrions comparer les performances observées lors du redémarrage partiel d'une application aux performances obtenues lors du redémarrage global depuis un point de reprise coordonné. À plus long terme, il serait important de trouver une solution pour pouvoir choisir et adapter les groupes de processus au cours de l'exécution de l'application. Si le graphe de communication des applications change au cours du temps, il faudrait alors concevoir un protocole hiérarchique capable de supporter des groupes dynamiques.

Remerciements

Les expériences présentées dans ce papier furent menées sur la plate-forme expérimentale Grid'5000, développée par le projet INRIA ALADDIN, avec le support du CNRS, de RENATER, de plusieurs universités ainsi que d'autres soutiens financiers (voir https ://www.grid5000.fr). Ces travaux ont été réalisés dans le cadre du Laboratoire commun INRIA-Illinois sur le calcul Petascale et le projet RESCUE et SPADES financés par l'ANR.

8. Bibliographie

Alvisi L., Marzullo K., « Message Logging : Pessimistic, Optimistic, Causal, and Optimal », *IEEE Transactions on Software Engineering*, vol. 24, n° 2, p. 149-159, 1998.

Alvisi L., Rao S., Husain S. A., Asanka d. M., Elnozahy E., « An Analysis of Communication-Induced Checkpointing », *FTCS '99 : Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, IEEE Computer Society, Washington, DC, USA, p. 242, 1999.

Antypas K., Shalf J., Wasserman H., NERSC-6 Workload Analysis and Benchmark Selection Process, Technical Report n° LBNL-1014E, Lawrence Berkeley National Laboratory, Berkeley, 2008.

Asanovic K., Bodik R., Catanzaro B. C., Gebis J. J., Husbands P., Keutzer K., Patterson D. A., Plishker W. L., Shalf J., Williams S. W., Yelick K. A., The Landscape of Parallel Computing Research : A View from Berkeley, Technical Report n° UCB/EECS-2006-183, University of California, Berkeley, 2006.

asc.llnl.gov, « The Sequoia Benchmarks », , http ://asc.llnl.gov/sequoia/benchmarks/, 2009.

Bailey D. H., Barszcz E., Barton J. T., Browning D. S., Carter R. L., Dagum L., Fatoohi R. A., Frederickson P. O., Lasinski T. A., Schreiber R. S., Simon H. D., Venkatakrishnan V., Weeratunga S. K., « The NAS parallel benchmarks—summary and preliminary results », *Supercomputing '91 : Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, ACM, New York, NY, USA, p. 158-165, 1991.

Bhargava B., Lian S.-R., « Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach », *Proceedings of the Seventh Symposium on Reliable Distributed Systems, 1988*, p. 3-12, oct, 1988.

Bhatia K., Marzullo K., Alvisi L., « Scalable causal message logging for wide-area environments », *Concurrency and Computation : Practice and Experience*, vol. 15, n° 10, p. 873-889, 2003.

Borg A., Baumbach J., Glazer S., « A Message System Supporting Fault Tolerance », *SIGOPS Operating Systems Review*, vol. 17, n° 5, p. 90-99, 1983.

Bouteiller A., Bosilca G., Dongarra J., « Redesigning the Message Logging Model for High Performance », *Concurrency and Computation : Practice and Experience*, vol. 22, p. 2196-2211, November, 2010.

Bouteiller A., Herault T., Bosilca G., Dongarra J., « Correlated Set Coordination in Fault Tolerant Message Logging Protocols », *in* E. Jeannot, R. Namyst, J. Roman (eds), *Euro-Par 2011 Parallel Processing*, vol. 6853 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 51-64, 2011a.

Bouteiller A., Hérault T., Bosilca G., Dongarra J. J., « Correlated Set Coordination in Fault Tolerant Message Logging Protocols », *Euro-Par (2)*, p. 51-64, 2011b.

Bouteiller A., Ropars T., Bosilca G., Morin C., Dongarra J., « Reasons for a Pessimistic or Optimistic Message Logging Protocol in MPI Uncoordinated Failure Recovery », *IEEE International Conference on Cluster Computing (Cluster 2009)*, New Orleans, USA, 2009.

Briatico D., Ciuffoletti A., Simoncini L., « A Distributed Domino Effect Free Recovery Algorithm », *IEEE International Symposium on Reliability, Distributed Softwares, and Databases*, p. 207-215, 1984.

Cappello F., « Fault Tolerance in Petascale/ Exascale Systems : Current Knowledge, Challenges and Research Opportunities », *International Journal of High Performance Computing Applications*, vol. 23, p. 212-226, August, 2009.

Cappello F., Guermouche A., Snir M., « On Communication Determinism in Parallel HPC Applications », *19th International Conference on Computer Communications and Networks (ICCCN 2010)*, 2010.

Chandy K., Lamport L., « Distributed Snapshots : Determining Global States of Distributed Systems », *ACM Transactions on Computer Systems*, vol. 3, n° 1, p. 63-75, 1985.

Daly J., « A model for predicting the optimum checkpoint interval for restart dumps », *Proceedings of the 2003 international conference on Computational science*, ICCS'03, Springer-Verlag, Berlin, Heidelberg, p. 3-12, 2003.

Daly J. T., « A higher order estimate of the optimum checkpoint interval for restart dumps », *Future Generation Computer Systems*, vol. 22, p. 303-312, February, 2006.

Damani O. P., Garg V. K., « How to Recover Efficiently and Asynchronously when Optimism Fails », *International Conference on Distributed Computing systems*, IEEE Computer Society, Los Alamitos, CA, USA, p. 108-115, 1996.

Elnozahy E. N. et al., System Resilience at Extreme Scale, Technical report, DARPA, 2008.

Elnozahy E. N. M., Alvisi L., Wang Y.-M., Johnson D. B., « A Survey of Rollback-Recovery Protocols in Message-Passing Systems », *ACM Computing Surveys*, vol. 34, n° 3, p. 375-408, 2002.

Elnozahy E. N., Zwaenepoel W., « Manetho : Transparent Roll Back-Recovery with Low Overhead, Limited Rollback, and Fast Output Commit », *IEEE Transactions on Computers*, vol. 41, n° 5, p. 526-531, 1992.

Gao Q., Huang W., Koop M. J., Panda D. K., « Group-based Coordinated Checkpointing for MPI : A Case Study on InfiniBand », *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, IEEE Computer Society, Washington, DC, USA, p. 47-, 2007.

Guermouche A., Ropars T., Brunet E., Snir M., Cappello F., « Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic Message Passing Applications », *25th IEEE International Parallel & Distributed Processing Symposium (IPDPS2011)*, Anchorage, USA, 2011.

Gupta B., Nikolaev R., Chirra R., « A Recovery Scheme for Cluster Federations Using Sender-based Message Logging », *Journal of Computing and Information Technology*, vol. 19, n° 2, p. 127-139, 2011.

Gupta B., Rahimi S., Allam V., Jupally V., « Domino-effect free crash recovery for concurrent failures in cluster federation », *Proceedings of the 3rd international conference on Advances in grid and pervasive computing*, GPC'08, Springer-Verlag, Berlin, Heidelberg, p. 4-17, 2008.

Ho J. C. Y., Wang C.-L., Lau F. C. M., « Scalable Group-Based Checkpoint/Restart for Large-Scale Message-Passing Systems », *22nd IEEE International Parallel and Distributed Processing Symposium*, Miami, USA, 2008.

Johnson D. B., Zwaenepoel W., « Sender-Based Message Logging », *Digest of Papers : The 17th Annual International Symposium on Fault-Tolerant Computing*, p. 14-19, 1987.

Kogge P. et al., Exascale Computing Study : Technology Challenges in Achieving Exascale Systems, Technical report, DARPA, 2008.

Komatitsch D., Tsuboi S., Ji C., Tromp J., « A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator », p. 4-11, 2003. Gordon Bell Prize winner article.

Koo R., Toueg S., « Checkpointing and Rollback-Recovery for Distributed Systems », *Proceedings of 1986 ACM Fall joint computer conference*, ACM '86, IEEE Computer Society Press, Los Alamitos, CA, USA, p. 1150-1158, 1986.

Lamport L., « Time, Clocks, and the Ordering of Events in a Distributed System », *Communications of the ACM*, vol. 21, n° 7, p. 558-565, 1978.

Lampson B. W., Sturgis H. E., Crash Recovery in a Distributed Data Storage System, Technical report, Xerox Palo Alto Research Center, 1979.

Meneses E., Mendes C. L., Kale L. V., « Team-based Message Logging : Preliminary Results », *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).*, May, 2010.

Message Passing Interface Forum, « MPI : A Message-Passing Interface Standard », , www.mpi-forum.org, 1995.

Monnet S., Morin C., Badrinath R., « Hybrid Checkpointing for Parallel Applications in Cluster Federations », *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID'04)*, IEEE Computer Society, Washington, DC, USA, p. 773-782, 2004.

Moody A., Bronevetsky G., Mohror K., Supinski B. R. d., « Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System », *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, IEEE Computer Society, Washington, DC, USA, p. 1-11, 2010.

Neves N., Fuchs W. K., « Using Time to Improve the Performance of Coordinated Checkpointing », *Proceedings of the International Computer Performance & Dependability Symposium*, p. 282-291, 1996.

Oldfield R. A., Arunagiri S., Teller P. J., Seelam S., Varela M. R., Riesen R., Roth P. C., « Modeling the Impact of Checkpoints on Next-Generation Systems », *MSST '07 : Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, IEEE Computer Society, Washington, DC, USA, p. 30-46, 2007.

Randell B., « System structure for software fault tolerance », *Proceedings of the international conference on Reliable software*, ACM, New York, NY, USA, p. 437-449, 1975.

Ropars T., Guermouche A., Uçar B., Meneses E., Kalé L. V., Cappello F., « On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications », *Euro-Par (1)*, p. 567-578, 2011a.

Ropars T., Morin C., « Fault Tolerance in Cluster Federations with O2P-CF », *Resilience 2008, Workshop on Resiliency in High Performance Computing*, Lyon France, 05, 2008.

Ropars T., Morin C., « Active Optimistic Message Logging for Reliable Execution of MPI Applications », *15th International Euro-Par Conference*, Delft, The Netherlands, p. 615-626, August, 2009.

Ropars T., Morin C., « Active Optimistic and Distributed Message Logging for Message-Passing Applications », *Concurrency and Computation : Practice and Experience*, vol. n/a, p. n/a-n/a, 2011b.

Sistla A. P., Welch J. L., « Efficient Distributed Recovery Using Message Logging », *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing (PODC '89)*, ACM Press, New York, NY, USA, p. 223-238, 1989.

Smith S. W., Johnson D. B., « Minimizing Timestamp Size for Completely Asynchronous Optimistic Recovery with Minimal Rollback », *Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS '96)*, IEEE Computer Society, Washington, DC, USA, p. 66, 1996.

Strom R. E., Yemini S., « Optimistic Recovery in Distributed Systems », *ACM Transactions on Computing Systems*, vol. 3, n° 3, p. 204-226, 1985.

Tamir Y., Séquin C. H., « Error Recovery in Multicomputers Using Global Checkpoints », *In International Conference on Parallel Processing*, p. 32-41, 1984.

Vaidya N., « Distributed Recovery Units : An Approach for Hybrid and Adaptive Distributed Recovery », *Technical Report 93-052, Departement of Computer Science Texas, A&M, University*, 1993.

www.top500.org, « TOP500 List of Worlds Supercomputers », , http ://www.top500.org/lists, June, 2011.

Yang J.-M., Li K. F., Li W.-W., Zhang D.-F., « Trading Off Logging Overhead and Coordinating Overhead to Achieve Efficient Rollback Recovery », *Concurrency and Computation : Practice and Experience*, vol. 21, p. 819-853, April, 2009.